16-bit CPU

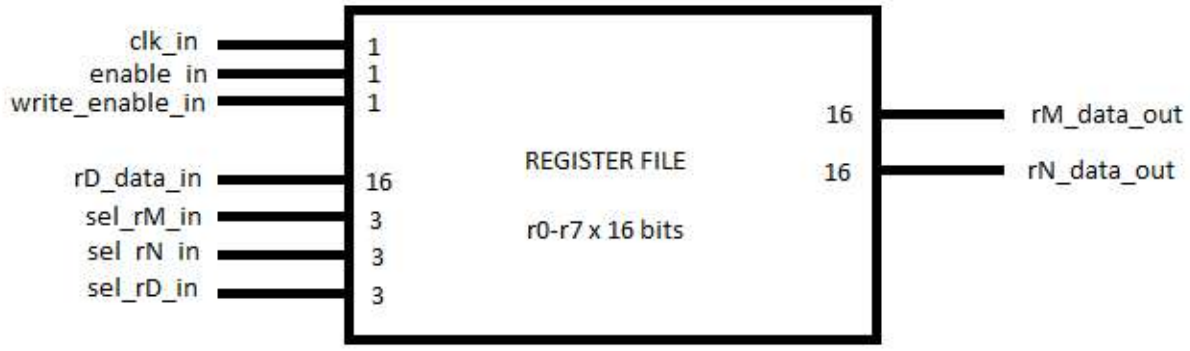
Alexander Archer

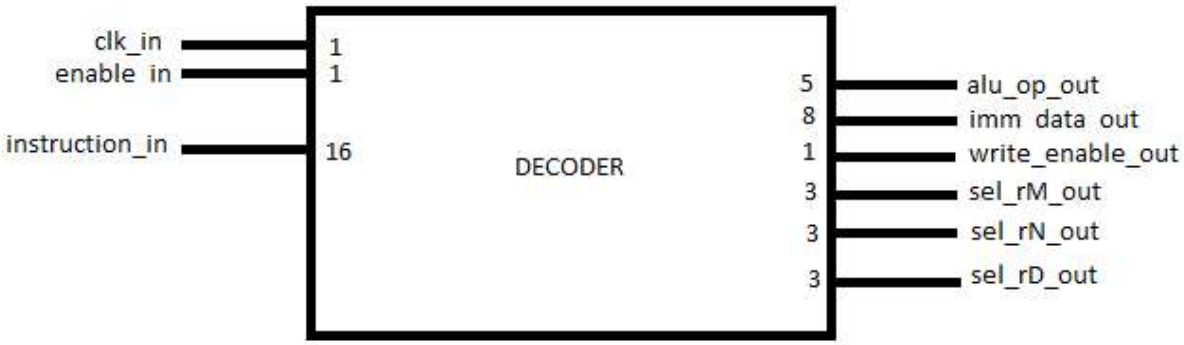
Overview

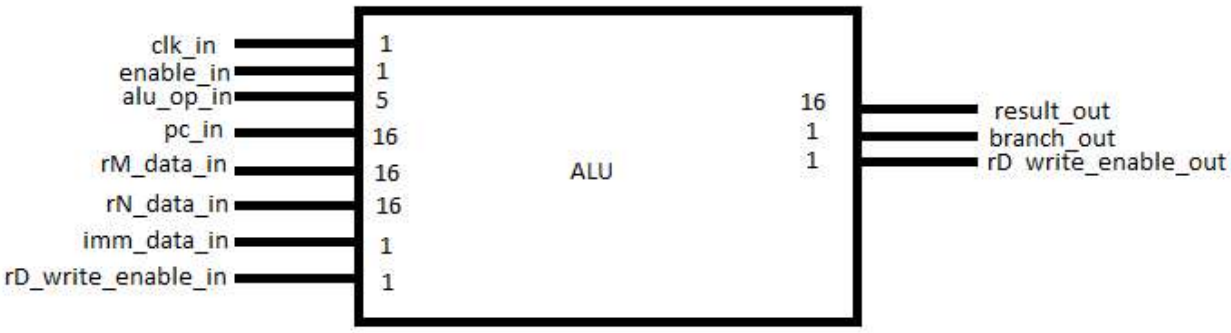
I wrote a simple but (mostly) complete 6-stage non-pipelined 16-bit CPU architecture that runs in Vivado simulation. This simulated CPU has 6 components: a register file, a decoder, an ALU, a control unit, a program counter, and ram/memory. Some of my inspiration for this project came from the ARM7 ISA which is the ISA I am the most familiar with; I have used a few of their naming conventions and some ISA layout. Included in this write up are the block diagrams of the components, the ISA layout, the instructions I implemented, and the shortcomings of the project.

The opcode section of the ISA has 4 bits, and I implemented 14 different instructions. 16 bits are extremely limiting, and therefore I was only able to fit 8 general purpose registers into the register file: RRR-type instructions are the largest, and take up 9 bits in just register numbers, and when including the opcode, 13. I decided to use 1 bit as a condition bit for various operations; therefore, at least 2 bits are unused in any given instruction. The simulated clock runs at a 10ns period or 100Mhz, and the simulated memory is 512 bytes. Given more time, I would have liked to have used the on-board memory and clock along with physical input/output to demonstrate and alter computation.

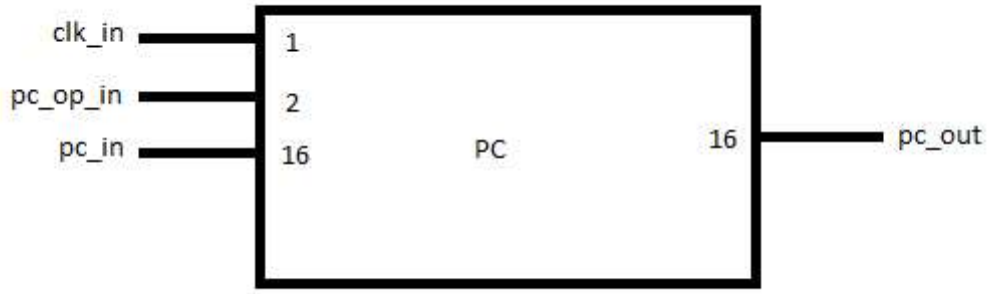
Component block diagrams

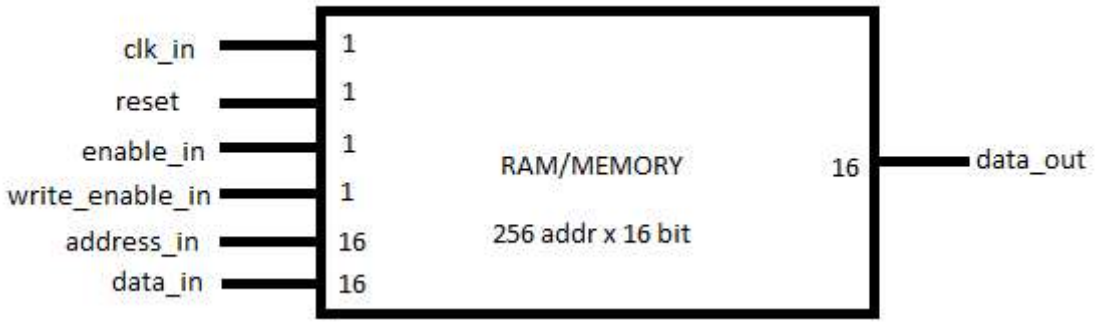



clk_in
enable_in
instruction_in
1
1
16
DECODER
5
8
1
3
3
3
alu_op_out
imm_data_out
write_enable_out
sel_rM_out
sel_rN_out
sel_rD_out


clk_in
enable_in
alu_op_in
pc_in
rM_data_in
rN_data_in
imm_data_in
rD_write_enable_in
1
1
5
16
16
16
1
1
ALU
16
1
1
result_out
branch_out
rD_write_enable_out


clk_in
reset_in
alu_op_in
1
1
5
Control Unit
6
stage_out


clk_in
pc_op_in
pc_in
1
2
16
PC
16
pc_out



Instructions

Terms:

R = register

U = unused

rD = destination register

rM = first operand register

rN = second operand register

c = condition bit

I(8)/(5) = immediate data bits, 8 or 5

Instruction	Form	Implementation	Condition bit	OPCODE
ADD	RRR	rD = rM + rN	c: 1/0 = signed/unsigned	0000
SUB	RRR	rD = rM - rN	c: 1/0 = signed/unsigned	0001
NOT	RRU	rD = not rN	c: N/A	0010
AND	RRR	rD = rM and rN	c: N/A	0011
OR	RRR	rD = rM or rN	c: N/A	0100
XOR	RRR	rD = rM xor rN	c: N/A	0101
LSL	RRI(5)	rD = rM << rN	c: N/A	0110
LSR	RRI(5)	rD = rM >> rN	c: N/A	0111
CMP	RRR	rD = cmp(rM, rN)	c: 1/0 = signed/unsigned	1000
B	UI(8)	PC = rM or 8-bit immediate	c: 1/0 = rM/8-bit immediate	1001
BEQ	URR	PC = rM conditional on rN	c: N/A	1010
IMMEDIATE	RI(8)	rD = 8-bit immediate	c: 1/0 = upper/lower 8-bits	1011
LD	RRU	rD = memory(rM)	c: N/A	1100
ST	URR	memory(rM) = rN	c: N/A	1101

Instruction Layout

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRR	C	opcode				rD			rM			rN			U	
RRU	C	opcode				rD			rM			U				
URR	C	opcode				U			rM			rN			U	
RRI(5)	C	opcode				rD			rM			5-bit imm				
RI(8)	C	opcode				rD			8-bit immediate							
UI(8)	C	opcode				U			8-bit immediate							

Selected instructions and waveform

The follow instructions are hardcoded into the memory, similar to what “rom” might be, for testing purposes; any instructions could be put there, but this set tests every instruction at least once.

Instructions: 0-11

```
'0' & "1011" & "000" & x"0A", -- imm r0 = 0x000A (580A)                                  0
'0' & "1011" & "001" & x"AA", -- imm r1 = 0x00AA (59AA)                                  1
'0' & "0010" & "001" & "001" & "00000", -- not r1 = xFF55 (1120)                         2
'0' & "0000" & "010" & "001" & "000" & "00", -- (unsigned) ADD r2 r1 r0, r2 = xFF5F (0220) 3
'0' & "1011" & "011" & x"08", -- imm r3 = 0x08  (5B08)                                   4
'0' & "0110" & "010" & "010" & "011" & "00", -- LSL r2 by r3, r2 = x5F00  (324C)         5
'0' & "1011" & "100" & x"04", -- imm r4 = 0x04       (5C04)                              6
'0' & "0111" & "010" & "010" & "100" & "00", -- LSR r2 by r4, r2 = x05F0 (3A50)          7
'1' & "1001" & "000" & x"0B", -- Branch to ram address 11 (C80B)                         8
x"0000", -- NOP (0000)                                                                   9
x"0000", -- NOP (0000)                                                                   10
'0' & "0001" & "101" & "010" & "011" & "00", -- (unsigned) SUB r5 r2 r3, r5 = x05E8 (0D4C) 11
```

Waveforms: 0-11

Instructions: 12-26

```
'0' & "1011" & "110" & x"FF", -- imm r6 = 0xFF       (5EFF)                                    12
'0' & "1101" & "000" & "110" & "101" & "00", -- ST r5 (rM) data at r6 (rN) address (x05E8 at xFF) (68D4)   13
'0' & "1100" & "111" & "110" & "00000", -- LD r6 address into r7, r7 = x05E8 (67c0)            14
'0' & "1000" & "000" & "101" & "111" & "00", --unsigned CMP r0 r5 r7, r0 = x"4000" (40BC)      15
'0' & "1011" & "001" & x"18", -- imm r1 = 0x18       (5918)                                    16
'0' & "1010" & "000" & "001" & "000" & "00",  -- BEQ r1 r0 .. r1 = addr, r0 = out from cmp (5020)   17
x"0000", -- nop (0000)                                                                         18
x"0000", -- nop (0000)                                                                         19
x"0000", -- nop (0000)                                                                         20
x"0000", -- nop (0000)                                                                         21
x"0000", -- nop (0000)                                                                         22
x"0000", -- nop (0000)                                                                         23
'0' & "0011" & "010" & "011" & "001" & "00", --AND r2 r3 r1, r2 = x08 (1A64)                   24
'0' & "0100" & "011" & "010" & "001" & "00", -- OR r3 r2 r1, r3 = 0x14 (2344)                  25
'0' & "0101" & "100" & "011" & "011" & "00", -- XOR r4 r3 r3, r4 = 0x00  (2C6C)                26
```

Waveforms: 12-26

The instruction images are labeled by instruction number and with the hexadecimal instruction value, seen in the ram_data_out and instruction lines on the waveforms.

The highlighted signals are the signals most worth noting, specifically the register data and select lines, the instruction line, the result line, the address line, and the ram lines.

The arithmetic and logical operators are fairly easy to validate: rD, rM, rN data and select lines are easily visible. The first interesting instruction is 8 which is a branch to memory location 11; we can verify this

worked correctly by observing the address line moving from x0008 to x000B, and the next two instructions (NOPS) in memory do not appear in the ram_data_out or instruction lines. The next interesting instructions are 13 and 14, a store and load, respectively; the store and load can be most easily observed by noting the ram_enable line going high in the cpu stages. The correctness here is validated by r5 and r7 containing the same data; this condition is then tested by instructions 15 and 17--a compare and branch if equal--which are correct given it does branch to memory location 24, again skipping the NOPS.

Shortcomings

- Only has 8 general purpose registers
- Can only load or store 16 bits at a time
- Signed overflows are possible and do not indicate when they occur
- I only implemented BEQ as a conditional branch
- The clock is a simulated clock
- The memory is a simulated memory, and only has one memory (ROM/RAM in same area)
- I did not test every iteration of every operation, but I tried at least one version of every type, therefore there are likely bugs that I did not catch during testing